TC2xx debug protection (with HSM)

AURIX™ 32-bit microcontrollers

About this document

Scope and purpose

AURIX™ TC2xx device debug protection is one of the implemented security layers applied by TriCore™ and the HSM Core to lock the debug interface in order to prevent unauthorized access from external tools. It also prevents unauthorized Flash read and write access from an external debugger as well. The debug access to the entire device can be enabled or disabled by AURIX™ TC2xx boot mode and by HSM itself as well as the Flash read and write protection.

This application note describes the AURIX™ TC2xx device debug protection with some possible use cases. Guidelines for the correct user configuration settings are discussed, and examples are shown to demonstrate how to lock and unlock the debug interface.

Intended audience

This document is intended for hardware and software engineers who need to setup AURIX™ TC2xx device debug protection with/without HSM enabled.

Abbreviations

Table 1

Abbreviation	Description
TC2xx	1st Generation AURIX™
HSM	Hardware Security Module
Host	In AURIX™, Host means TriCore™ because it is the host of the HSM
PMU0	Program Memory Unit 0
UCB	User Configuration Block
SSW	Startup Software
OCDS	On Chip Debug Support
PROCON	Protection configuration register
DF_UCB	Data Flash UCB
OEC	OCDS enable control register
OSCU	OCDS System Control Unit

References

[1] HSM Target Specification – Rev 1.4, 2012-07-31. See document at **www.infineon.com**

[2] TC23x/TC27x/TC29x User’s Manual – V1.0, 2014-01 or later. See document at **www.infineon.com**

Table of contents

About this document ... **1**
Table of contents ... **2**
1 AURIX™ TC2xx device debug protection ... **3**
1.1 Lock conditions for the debug interface protection mechanism ... 3
1.2 OCDS System Control Unit ... 3
1.2.1 Enabling OCDS ... 4
1.3 Internal gates for debug protection ... 5
1.4 Configuration registers for debug protection ... 6
1.4.1 User Configuration Block confirmation ... 8
1.4.2 UCB_DBG to lock and unlock TriCore™ debug access ... 9
1.4.3 UCB_HSM to lock and unlock TriCore™ and HSM debug access ... 10
1.4.4 UCB_PFLASH/UCB_DFLASH to enable or disable Read protection ... 13
1.5 Flash command sequence definitions ... 14
1.5.1 Command sequence overview ... 15
1.5.2 Debug Protection Disable Status ... 16
1.6 SSW debug handling to unlock the debug interface ... 16
2 Debug access protection use-cases ... **19**
2.1 Use-case 1: TriCore™ configure its own debug interface protection ... 19
2.2 Use-case 2: TriCore™ configure its own debug interface protection, with Flash read protection ... 20
2.3 Use-case 3: TriCore™ and HSM configure their own debug access protection separately ... 21
2.4 Use-case 4: TriCore™ and HSM configure their own debug access protection, with Flash read protection ... 23
2.5 Use-case 5: HSM configures both TriCore™ and HSM Debug access protection, with Flash read protection ... 25
2.6 Use-case 6: HSM configures both TriCore™ and HSM debug access protection, with Debug password protection ... 26
3 Lock/unlock the debug access using debugger tools ... **27**
3.1 Lock using the Infineon Memtool/PLS UDE Memtool ... 27
3.2 Unlock using the Infineon Memtool/PLS UDE Memtool ... 29
3.3 Lock using Lauterbach ... 29
3.4 Unlock using Lauterbach ... 30
3.5 Locking using iSystem ... 30
3.6 Unlocking using iSystem ... 33
4 Lock/unlock debug interface using software ... **34**
4.1 Enable debug access password protection using software ... 34
4.2 Disable protection mechanism for UCB_DBG using software ... 36
4.3 Resume protection mechanism for UCB_DBG using software ... 36
4.4 Clear OCDS Interface Locked indication bit using software ... 36
Revision history ... **38**

1 AURIX™ TC2xx device debug protection

1.1 Lock conditions for the debug interface protection mechanism

Debug access is a combination of the OCDS system and the debug interface lock. The debug access to the entire device (TriCore™ + HSM) can be prevented using one of the following lock conditions:

- Disabling the OCDS System via the PROCONDBG.OCDSDIS bit in the UCB_DBG.
- Locking the TriCore™ debug interface using the PROCONDBG.DBGIFLCK bit with a Password Protection mechanism in UCB_DBG or the PROCONHSM.DBGIFLCK bit in the UCB_HSM.
- Locking the TriCore™ debug interface with Flash Read Protection, using the PROCONPF.RPRO in the UCB_PFLASH or the PROCONDF.RPRO in the UCB_DFLASH.
- Disabling the HSM debug access via the PROCONHSM.HSMDBGDIS in the UCB_HSM.

These lock conditions are checked and applied during the startup boot by the SSW.

Note: *When the OCDS is disabled using the PROCONDBG.OCDSDIS bit in the UCB_DBG, this applies only for TriCore™. The HSM debug support is not impacted.*

Debug Protection Mechanism	Locked	Unlocked
OCDS	• **PROCONDBG.OCDSDIS** in UCB_DBG	➔With **correct debug interface password** by SSW or by Application SW (disable protection)
TriCore™	• **PROCONDBG.DBGIFLCK** in UCB_DBG	➔With **correct debug interface password** by SSW or by Application SW (disable protection)
TriCore™	With any Flash read protection is configured • **PROCONPF.RPRO** in UCB_PFLASH • **PROCONDF.RPRO** in UCB_DFLASH	➔**OSTATE.IF_LCK** OCDS register need to be changed by SSW or by Application SW (OCDS_clear_interface_locked)
HSM Core	• **PROCONHSM.DBGIFLCK** in UCB_HSM • **PROCONHSM.HSMDBGDIS** in UCB_HSM	➔**DBGCTRL.HOST** ➔**DBGCTRL.HSM** HSM Debug Control register need to be changed by Application SW in HSM Core

Figure 1 Overview of locked and unlocked debug protection

1.2 OCDS System Control Unit

The aim of the OCDS System Control Unit (OSCU) is to control the OCDS features on SoC designs in a standardized way.

The features it controls include:

- The OCDS enabling co-operation with the HSM
- The Halt After Reset (HARR)
- The key mechanism to enable the OCDS by software in a secure way
- The Hot attach of a tool to a running system

1.2.1 Enabling OCDS

The OCDS is usually disabled for the following reasons:

- To prevent unintended effects from the OCDS system if no tool is present.
- To disable OCDS resources for power saving reasons

A robust OCDS enabling mechanism is required in order to enable or disable the OCDS in a proper and secure way.

The following picture shows how the OCDS state machine works to activate the debug access in a controlled manner.

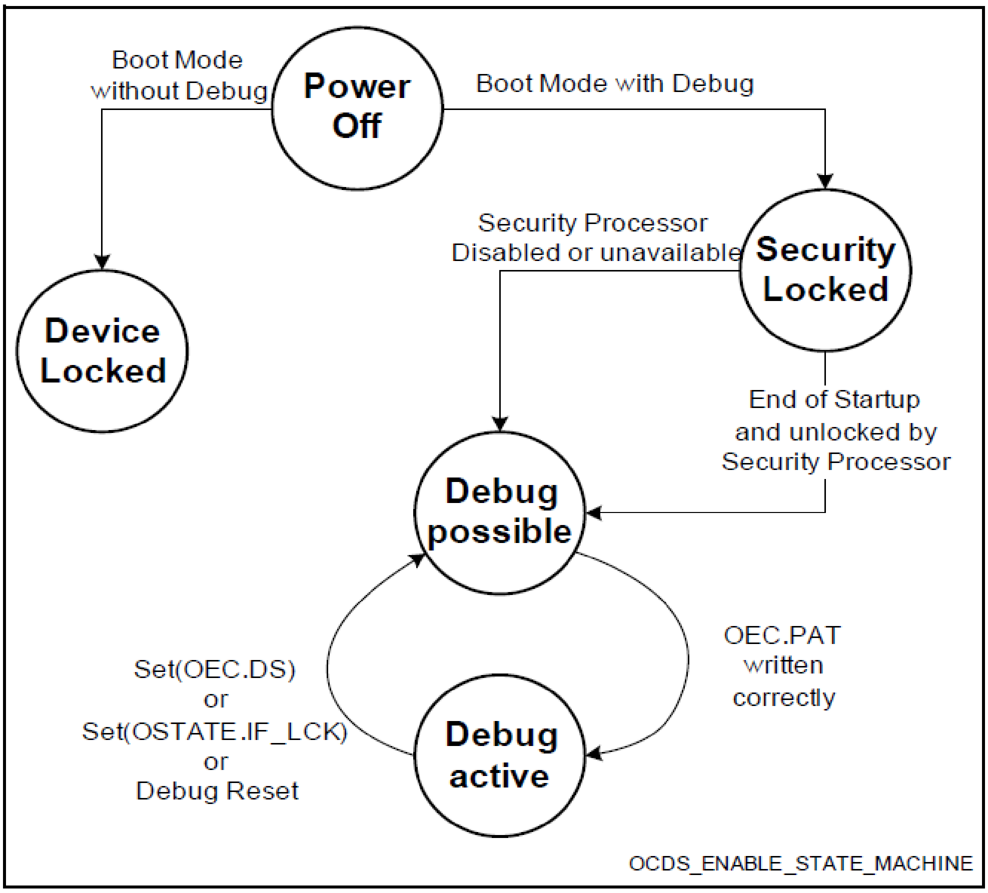


Figure 2 Overview on OCDS state machine

At the boot there can be only two available states:

- **Device locked**
  - All debug resources, including break and suspend logic for example, are turned off and no access via the IOClient is possible.
  - Cerberus is in power saving mode (OSTATE.OEN is 0_B by HW).
- **Security locked**
  - All the debug resources are disabled (OSTATE.OEN is 0_B).

In the Security Locked state the Communication Mode (COM mode) via IOClient can be used to communicate with the HSM. It is not possible to enter in the Read/Write Mode (RW mode) unless the SSW has finished and the HSM agrees. Then:

- **Debug possible**
  - All the debug resources are disabled.
  - The COM mode can be used via IOClient but the RW mode access is only possible when the OSTATE.IF_LCK is cleared.
- **Debug active**
  - All the debug resources are available.
  - Unlimited RW mode access is possible when OSTATE.IF_LCK is cleared and unlimited access to debug resources via system bus is possible.

In order to pass from the **Debug possible** state to the **Debug active** state, it is necessary to write a correct pattern using the OEC.PAT register (see **4.4**).

This protection scheme can be considered as a combination of the Gates 1 and 3 (see **1.3**).

1.3 Internal gates for debug protection

The debug interface protection can be explained with the following diagram, where several gates have been put in place to clarify the progressive security layer mechanism applied:

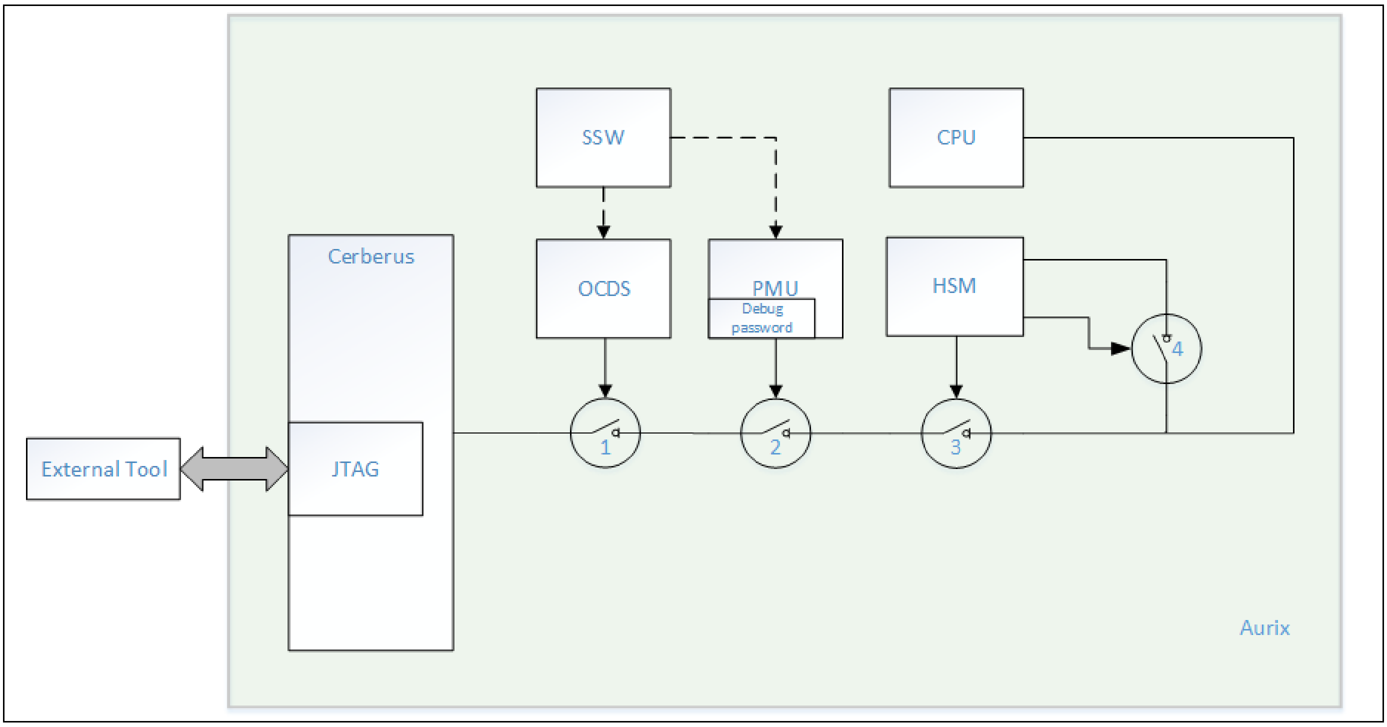


Figure 3 **Overview of gates used to lock and unlocked the debug access**

- Gate 1 controls the access to the TriCore™ debug interface via the OCDS module.
- Gate 2 controls the access to the TriCore™ debug interface via the PMU module.
- Gates 3 and 4 are directly controlled by the HSM. With these two gates, the HSM can lock the debug access to the Host CPU and also disable the debug support for the HSM itself. The locking mechanism is controlled by the UCB_HSM.

1.4 Configuration registers for debug protection

AURIX™ TC2xx devices provide several security protection layers in order to restrict the debugger access to the entire microcontroller.

The configuration of these layers is based on the User Configuration Blocks (UCBs) which are specific logical sectors DF_UCB (Data Flash UCB), a sub-sector of bank DF0 (Data Flash 0), inside the PMU0 module (Program Memory Module 0).

- Most of the UCBs contain protection settings and other parameters that are configurable by the user.
- The DF_UCB contains 16 logical sectors, UCB0 to UCB15, with 1Kbyte each.

In the address map of the PMU0, the DF_UCB starts from the address 0xAF100000 to 0xAF103FFF, with a size of 16Kbyte.

Start Address	Size	Range	Start Address Symbol
8000 0000$_H$	2 MByte	PF0	AC_PF0
8020 0000$_H$	2 MByte	PF1	AC_PF1
8FFF 8000$_H$	32 KByte	BootROM	AC_BROM0
A000 0000$_H$	2 MByte	PF0	AN_PF0
A020 0000$_H$	2 MByte	PF1	AN_PF1
AF00 0000$_H$	384 KByte	DF0 (DF_EEPROM)[1]	AN_DFlash_B0
AF10 0000$_H$	16 KByte	DF0 (DF_UCB)	
AF11 0000$_H$	64 KByte	DF1[2]	AN_DFlash_B1
AFFF 8000$_H$	32 KByte	BootROM	AN_BROM0
F800 0500$_H$	256 Byte	PMU Registers	
F800 1000$_H$	5 KByte	Flash Registers	
FF11 0000$_H$	64 KByte	DF1 (HSM private access)[2]	AN_DFlash_B1F

Figure 4 Address map of PMU0

The following figure gives a detailed view of the DF_UCB sector structure inside the DFlash Bank, which shows all 16 UCBs used for the protection mechanisms.

Logical Sector	Log. Sub-Sector	Size	Offset Address
UCB0 = UCB_PFlash	DF_UCB	1 KByte	10'0000$_H$
UCB1 = UCB_DFlash		1 KByte	10'0400$_H$
UCB2 = UCB_HSMCOTP		1 KByte	10'0800$_H$
UCB3 = UCB_OTP		1 KByte	10'0C00$_H$
UCB4 = UCB_IFX		1 KByte	10'1000$_H$
UCB5 = UCB_DBG		1 KByte	10'1400$_H$
UCB6 = UCB_HSM		1 KByte	10'1800$_H$
UCB7 = UCB_HSMCFG		1 KByte	10'1C00$_H$
UCB8		1 KByte	10'2000$_H$
UCB9		1 KByte	10'2400$_H$
UCB10		1 KByte	10'2800$_H$
UCB11		1 KByte	10'2C00$_H$
UCB12		1 KByte	10'3000$_H$
UCB13		1 KByte	10'3400$_H$
UCB14		1 KByte	10'3800$_H$
UCB15		1 KByte	10'3C00$_H$

Figure 5 DF_UCB structure

- The effective protection applied via the UCBs is determined by the content of the Protection Configuration, PROCONx registers. (I.e. PROCONDBG, PROCONHSM, etc.)
- The PROCONx registers are located in the FLASH0 module of the PMU0 starting from the address 0xF8002020.
- The PROCONx registers are loaded automatically by hardware during device startup, after each reset, from the UCB locations.
- The UCB is always evaluated during device startup.

Note:

1. *In this application note, only the UCB_HSM, the UCB_DBG, UCB_PFLASH and UCB_DFLASH are discussed, as they are the only registers required to lock the debug access to the HSM and TriCore™ and protect the flash location from external tools.*
2. *Please refer to the AURIX™ TC2xx User Manual in order to acquire more information regarding the remaining UCBs and PROCONx registers not explicitly covered in this document. [2].*

1.4.1 User Configuration Block confirmation

The User Configuration Blocks (UCBs), or multi-sets of a UCB in the case of UCB_OTP and UCB_HSMCOTP, can be in the states:

- CONFIRMED
- UNLOCKED
- ERASED
- ERRORED

The state is determined by the content of the confirmation code:

- 0x57B5327F = the state is CONFIRMED, resulting in the UCB read/write access being restricted
- 0x43211234 = the state is UNLOCKED, resulting in the UCB read/write access being allowed
- 0x00000000 = the state is ERASED, resulting in the same behavior as in ERRORED state, which means that the device will not boot anymore
- For all other value including uncorrectable ECC errors, the state is ERRORED, resulting in the device not booting anymore

Note: Any UCB which contains a password could have its UCB content changed even in the CONFIRMED State. This could be done by providing a correct password, with a flash disable protection command sequence.

Note: The effect of a new UCB content, such as a configuration including confirmation code for example, will only be evaluated and taken after the next reset.

ERRORED state

A UCB is also in its ERRORED state when at least one of its entries has an uncorrectable error in the original and its copy.

If during evaluation of the UCBs an ERRORED state is detected, the Protection Error flag FSR.PROER will be set. In this case all the protections related to the PROCONx register, which is associated to the UCB in the ERRORED state, are fully activated.

During startup, an errored UCB detected by Startup Software (SSW) is realized as a failed flash startup and consequently the SSW does not boot the device.

As the ERASED state is also considered as ERRORED, the transition from the UNLOCKED to CONFIRMED state can be done without erasing the UCB. For this the UNLOCKED value (0x43211234) in the pages with the confirmation code and the copied confirmation code can be over-programmed with the CONFIRMED value (0x57B5327F).

Note: In order to avoid ECC errors, the location at offset 0x74 and 0x7C, 4 bytes following the confirmation code, must not be modified from the original value (0x00000000) in the UNLOCKED state and in the over-programmed data.

CONFIRMED state

A UCB is in its CONFIRMED state when at least one of its entries has a CONFIRMED value in the original or its copy.

Note: The default password is 256 bit of zeros, if no password value has been entered on CONFIRMED state. (0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000).

1.4.2 UCB_DBG to lock and unlock TriCore™ debug access

The UCB_DBG configures the user defined debug password protection for the debug interface. It is Read/Write protected with the password PW0 to PW7.

When the UCB is confirmed by UCB confirmation code, the debug interface can be unlocked by supplying the correct password of UCB_DBG to the PMU (which is the same PW0 to PW7 password), under the condition that HSM has kept this interface unlocked.

UCB_DBG Content

Offset[1]	Content	Description
00_H	PROCONDBG	Protection of the debug interface.
10_H	PROCONDBG	Copy.
20_H	PW0 – PW7	256-bit password, from least significant word to most significant word (8 words).
40_H	PW0 – PW7	Copy of 256-bit password.
70_H	Confirmation	4 bytes.
78_H	Confirmation	Copy.

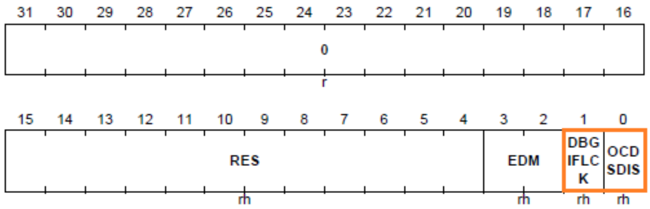
1) The offsets are given in number of bytes.

UCB_DBG Access Protection

This UCB is write and read protected if:

- (FPRO.PROINDBG and not FPRO.PRODISDBG) or (PROCONHSMCOTP.DESTDBG = "destructive" and (FDEST = 0 or "not executing Startup Software")) or PROCONDBG.EDM = "debug entered".

PROCONDBG
Debug Interface Protection Configuration (1058$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
OCDSDIS	0	rh	**OCDS Disabled** This bit indicates whether the OCDS is configured as locked. 0_B No OCDS lock configured in UCB_DBG. 1_B OCDS lock configured in UCB_DBG.
DBGIFLCK	1	rh	**Debug Interface Locked** This bit indicates whether the debug interface is configured as locked. 0_B No debug interface lock configured in UCB_DBG. 1_B Debug interface lock configured in UCB_DBG.

Figure 6 Layout of UCB5 (UCB_DBG)

The PROCONDBG, Debug Protection Configuration register has two bits related to the debug protection:

- Bit 0:
  - If set to 1_B, this bit indicates that the On Chip Debug Support is configured as locked. No debug support is allowed. All debug resources, such as breakpoints and suspend logic for example, are disabled.
- Bit 1:
  - If set to 1_B, this bit indicates that the debug interface is configured as locked.

Note:

1. *The UCB_DBG controls the "Debug password" gate. This UCB can be considered one of the first debug protection layers which an external tool has to manage in order to have access granted. If the DBGIFLCK bit is set to 1_B in the PROCONDBG register, the SSW will automatically unlock the debug interface only if a correct debug interface password is provided.*
2. *In the case where only the OCDSDIS flag is set to 1_B and DBGIFLCK flag is clear to 0_B in the PROCONDBG register, the SSW will not handle the unlocking of the debug interface even when a correct debug interface password is provided. The user will have to perform the flash disable protection command sequence in the TriCore™ application code.*

It is possible to change the content of the UCB_DBG as long as the confirmation code in the original or its copy is in the UNLOCKED state. When the UCB is in CONFIRMED state, it is possible to change the content of the UCB_DBG with the correct debug interface password (PW0-PW7).

Note:

3. *If both the DBGIFLCK bit and OCDSDIS bit are set to 1_B in the PROCONDBG register, the debug interface will be locked. By providing a correct debug interface password, this will unlock the debug interface as well as the read and write protection of the UCB. The UCB content will then be accessible, regardless of whether it is in the UNLOCKED or CONFIRMED state.*
4. *Where both the DBGIFLCK bit and OCDSDIS bit are cleared to 0_B and UCB_DBG is in CONFIRMED state, the debug interface will not be locked but the UCB read and write access will be restricted. User will have to perform the disable protection command sequence with a correct debug password in the application code to temporarily disable the UCB access protection.*

1.4.3 UCB_HSM to lock and unlock TriCore™ and HSM debug access

The UCB_HSM configures the user defined debug protection for the entire device. It is used to control both the HSM debug support and Chip debug interface, enabling or disabling the access for an external debugging tool.

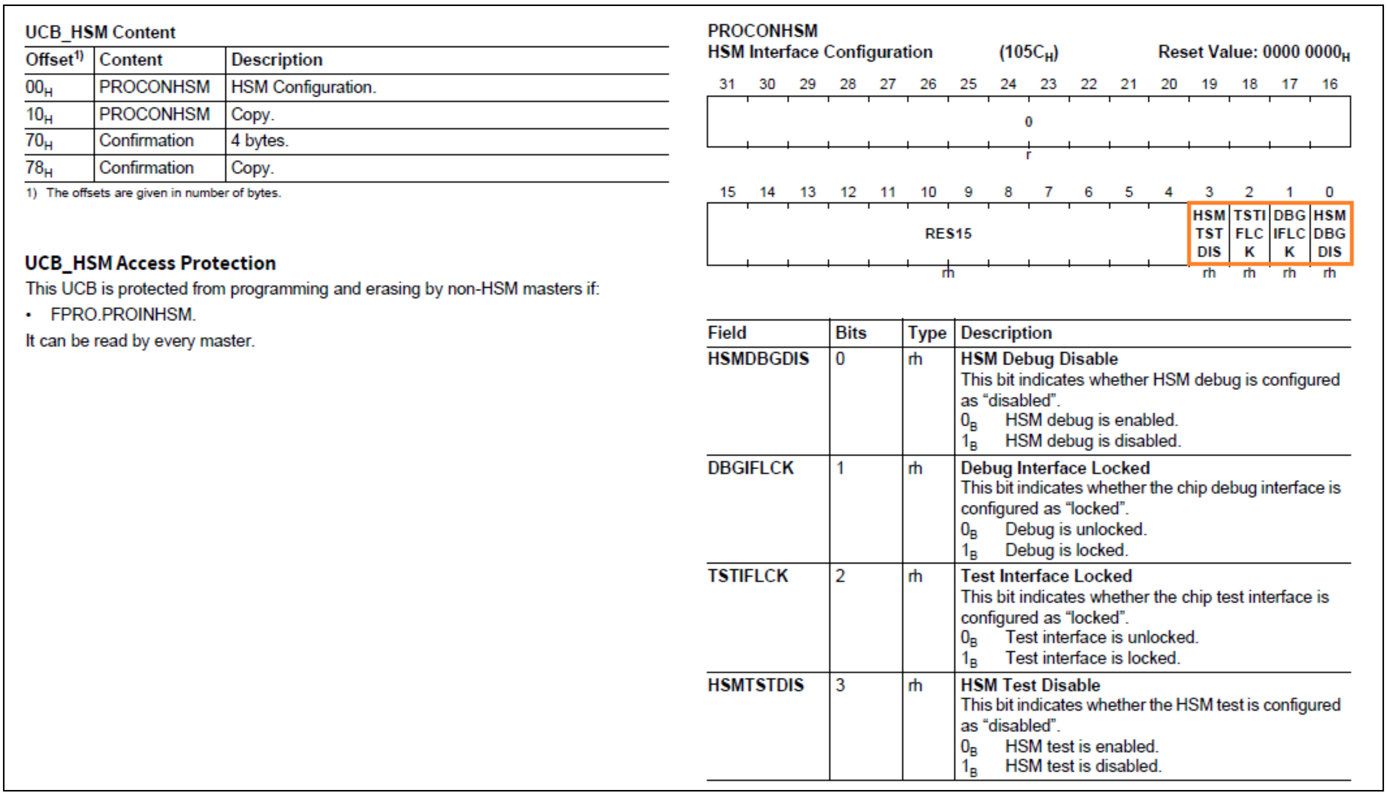
When the UCB is in CONFIRMED state, only the HSM can modify it.

The contents of UCB_HSM are only used by the device when HSM is enabled.

The protection configuration is transferred into the register PROCONHSM.

Note: In order to enable the HSM, the PROCONHSMCOTP.HSMBOOTEN bit in the UCB_HSMCOTP has to set to 1_B.

AURIX™ TC2xx device debug protection



UCB_HSM Content

Offset[1]	Content	Description
00_H	PROCONHSM	HSM Configuration.
10_H	PROCONHSM	Copy.
70_H	Confirmation	4 bytes.
78_H	Confirmation	Copy.

1) The offsets are given in number of bytes.

UCB_HSM Access Protection

This UCB is protected from programming and erasing by non-HSM masters if:

- FPRO.PROINHSM.

It can be read by every master.

PROCONHSM
HSM Interface Configuration ($105C_H$) **Reset Value: 0000 0000$_H$**

Field	Bits	Type	Description
HSMDBGDIS	0	rh	**HSM Debug Disable** This bit indicates whether HSM debug is configured as "disabled". 0_B HSM debug is enabled. 1_B HSM debug is disabled.
DBGIFLCK	1	rh	**Debug Interface Locked** This bit indicates whether the chip debug interface is configured as "locked". 0_B Debug is unlocked. 1_B Debug is locked.
TSTIFLCK	2	rh	**Test Interface Locked** This bit indicates whether the chip test interface is configured as "locked". 0_B Test interface is unlocked. 1_B Test interface is locked.
HSMTSTDIS	3	rh	**HSM Test Disable** This bit indicates whether the HSM test is configured as "disabled". 0_B HSM test is enabled. 1_B HSM test is disabled.

Figure 7 Layout of UCB6 (UCB_HSM)

The PROCONHSM, HSM Interface Protection Configuration register has two bits related to debug protection:

- Bit 0:
  - If set to 1_B, this bit indicates that the HSM debug support is configured as "disabled".
    - The term "disabled" refers to HSM debug support.
- Bit 1:
  - If set to 1_B, this bit indicates that the chip debug interface is configured as "locked".
    - The term "locked" refers to the AURIX™ device debug interface.

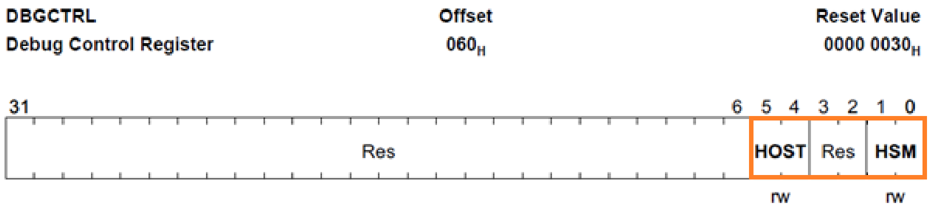
Note: Bits 2 and 3 are not used in AURIX™ and they can be programmed to 1_B or remain at 0_B.

Setting the PROCONHSM.HSMDBGDIS bit to 1_B disables the HSM debug support. Any access request to the HSM side from the host CPU or an external debugger tool will be restricted. Disabling HSM debug will not impact the debugger access to TriCore™. The user will only be able to debug the TriCore™ side.

Attention: In the case where the HSMDBGDIS bit is cleared to 0_B in the PROCONHSM register, HSM debugging is enabled. This will also allow the host CPU to access (read and write) the HSM internal resources through a 64-Kbyte window in the host system address space. For security reasons, a user shall program the HSMDBGDIS bit to 1_B on an AURIX™ with a HSM device, to ensure the HSM debug access is protected properly.

It is possible to change the content of the UCB_HSM as long as the confirmation code and its copy are in the UNLOCKED state. This means that when a user decides to move from the UNLOCKED state to the CONFIRMED state, the content of the UCB_HSM will not be changeable anymore from TriCore™. However, when its content is confirmed, only the HSM can modify this UCB_HSM.

Note: *If the UCB_HSM is set as CONFIRMED and the PROCONHSM.HSMDBGDIS bit is set to 1_B, leads to the condition that only the HSM can temporarily re-enable the debugger access for the HSM until the next reset. This can be done by the HSM using a register called DBGCTRL (Debug Control register), as described in the AURIX™ TC2xx HSM_TS_V1.0_Rev_1.4[1].*



Field	Bits	Type	Description
HOST	5:4	rw	**Debug control for host system** *Note: To enable host debugging, these bits must be set as described below and, in addition, the debug lock bit must be cleared, as explained in the Cerberus module description. See [10].* 00_B **HTDDIS0** Host debug disabled. 01_B **HTDDIS1** Host debug disabled. 10_B **HTDDIS2** Host debug disabled. 11_B **HTDEN** Host debug enabled.
HSM	1:0	rw	**Debug control for HSM subsystem and debug memory window** 00_B **HSMDDIS0** HSM debug disabled. 01_B **HSMDDIS1** HSM debug disabled. 10_B **HSMDDIS2** HSM debug disabled. 11_B **HSMDEN** HSM debug enabled.

Figure 8 Overview of the DBGCTRL register

1.4.4 UCB_PFLASH/UCB_DFLASH to enable or disable Read protection

The UCB_PFLASH and the UCB_DFLASH configure the read and write protection for the Program Flash and Data Flash memory locations. It is protected with the password PW0 to PW7. This is to restrict the possibility to write and read these areas from any external tool.

UCB_PFLASH

The protection configuration is transferred into the register PROCONPp (with "p" looping over the implemented Program Flash banks).

UCB_PFlash Content

Offset[1]	Content	Description
00_H	PROCONP0	Read protection for complete PFlash. Write protection for logical sectors of PF0.
04_H	PROCONP1	Write protection for logical sectors of PF1.
08_H	PROCONP2	Write protection for logical sectors of PF2.
$0C_H$	PROCONP3	Write protection for logical sectors of PF3.
10_H	PROCONP0	Copy.
14_H	PROCONP1	Copy.
18_H	PROCONP2	Copy.
$1C_H$	PROCONP3	Copy.
20_H	PW0 – PW7	256-bit password, from least significant word to most significant word (8 words).
40_H	PW0 – PW7	Copy of 256-bit password.
70_H	Confirmation	4 bytes.
78_H	Confirmation	Copy.

1) The offsets are given in number of bytes.

UCB_PFLASH Access Protection

This UCB is write and read protected if:

- FPRO.PROINP and not FPRO.PRODISP.

PROCONPp (p=0-3)
PFlash Protection Configuration PFp (1020_H+p*4_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RPRO	RES				S26L	S25L	S24L	S23L	S22L	S21L	S20L	S19L	S18L	S17L	S16L
rh	rh				rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S15L	S14L	S13L	S12L	S11L	S10L	S9L	S8L	S7L	S6L	S5L	S4L	S3L	S2L	S1L	S0L
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
RPRO	31	rh	**Read Protection Configuration** This bit indicates whether read protection is configured for PFLASH. This bit is only used in PROCONP0. In other PROCONPs it is reserved. 0_B No read protection configured 1_B Read protection and global write protection is configured.

Figure 9 Layout of UCB0 (UCB_PFLASH)

The PROCONP, PFlash Protection Configuration PF0/1 register mirrors the configuration set from the UCB_PFlash.

- When UCB_PFlash is not in ERRORED state, the PROCONP0/1 will indicate which PFlash sector is write protected or read protected.
- When UCB_PFlash is in ERRORED state, the device will not start.

The granularity of the write protection is provided by sectors. The sectors S6, S16 and S17, which are related to the HSM memory space, can be write-protected only if they are not flagged as HSM_exclusive.

The global read protection can be applied only for the Program Flash module in PROCONP0 register and at the same time it covers a global write protection.

Note: If the read protection with global write protection is configured, then the configured sector specific write protection will be overridden.

UCB_DFLASH

The protection configuration is transferred into the register PROCOND.

Table 10-10 UCB_DFlash Content

Offset[1]	Content	Description
00_H	PROCOND	Read and write protection for the DFlash sectors EEPROMs, oscillator configuration, and enable of RAM initialization.
10_H	PROCOND	Copy.
20_H	PW0 – PW7	256-bit password, from least significant word to most significant word (8 words).
40_H	PW0 – PW7	Copy of 256-bit password.
70_H	Confirmation	4 bytes.
78_H	Confirmation	Copy.

1) The offsets are given in number of bytes.

UCB_DFLASH Access Protection

This UCB is write and read protected if:

- FPRO.PROIND and not FPRO.PRODISD.

PROCOND
DFlash Protection Configuration (1030_H) Reset Value: $0000\ 0000_H$

31	30	29–28	27–16
RPRO	RES30	RES29	ESR0CNT
rh	rh	rh	rh

15	14	13	12	11	10–9	8	7–4	3–2	1	0
CAP3EN	CAP2EN	CAP1EN	CAP0EN	APREN	MODE	OSCCFG	RAMINSEL	RAMIN	NSAFECC	L
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
RPRO	31	rh	**Read Protection Configuration** This bit indicates whether read protection is configured for DFlash sectors EEPROMx. 0_B No read protection configured 1_B Read protection and write protection is configured.

Figure 10 Layout of UCB1 (UCB_DFLASH)

The PROCOND, DFlash Protection Configuration register exists only in PMUs with DFlash. It mirrors the configuration set from the UCB_DFlash. When the UCB_DFlash is in an ERRORED state, the device will not start.

UCB_PFLASH and UCB_DFLASH

Both registers have 1 bit related to the read protection mechanism:

- Bit 31
  - If set to 1_B, this bit indicates the read and write protection is enabled.

Note: When any Flash read protection is applied (PROCONP0.RPRO or PROCOND.RPRO are 1_B), the SSW leaves the debug interface locked (OSTATE.IF_LCK stays 1_B).

1.5 Flash command sequence definitions

The UCBs can be programmed and erased by all CPUs. The mechanism for how to perform these operations is described here.

The parameter “Address Data” used in the Command Sequence can be one of the following:

- PA
  - Absolute start address of the Flash page.
  - Must be aligned to burst size for “Write Burst” or to the page size for “Write Page”
- SA
  - Absolute start address of a Flash sector.
  - Allowed are the PFlash sectors Sx and the DFlash sectors HSMx, EEPROMx, UCBx.
- UC
  - Identification of the UCBx for which the password checking shall be performed:
  - $xx00_H$ for UCB0 = UCB_PFlash, to disable global read and sector specific write protection for all PFLASHs.
  - $xx01_H$ for UCB1 = UCB_DFlash, to disable global read and write protection for DFLASH sectors EEPROMs.
  - $xx05_H$ for UCB5 = UCB_DBG, to disable debug interface protection.

- PWx
  - 32-bit word of a 256-bit password.

Note: More details of Flash Command Sequence Definition are in the AURIX™ TC2xx User Manual[2].

1.5.1 Command sequence overview

In the following picture all the available command sequences are described, but only some command sequences are taken into account for the purposes of this application note.

Command Sequence		1. Cycle	2./6. Cycle	3./7. Cycle	4./8. Cycle	5./9. Cycle	6. Cycle
Reset to Read	Address	.5554					
	Data	..xxF0					
Enter Page Mode	Address	.5554					
	Data	..xx5y					
Load Page	Address	.55F0					
	Data	1) WD					
Write Page	Address	.AA50	.AA58	.AAA8	.AAA8		
	Data	PA	..xx00	..xxA0	..xxAA		
Write Page Once	Address	.AA50	.AA58	.AAA8	.AAA8		
	Data	PA	..xx00	..xxA0	..xx9A		
Write Burst	Address	.AA50	.AA58	.AAA8	.AAA8		
	Data	PA	..xx00	..xxA0	..xx7A		
Erase Logical Sector Range	Address	.AA50	.AA58	.AAA8	.AAA8		
	Data	SA	..xxnn	..xx80	..xx50		
Verify Erased Logical Sector Range	Address	.AA50	.AA58	.AAA8	.AAA8		
	Data	SA	..xxnn	..xx80	..xx5F		
Resume Prog/Erase	Address	.AA50	.AA58	.AAA8	.AAA8		
	Data	PA/SA	..xxnn	..xx70	..xxCC		
Clear Status	Address	.5554					
	Data	..xxFA					

Command Sequence		1. Cycle	2./6. Cycle	3./7. Cycle	4./8. Cycle	5./9. Cycle	6. Cycle
Disable Protection	Address	.553C	.553C	.553C	.553C	.553C	
	Data	UC	PW0/4	PW1/5	PW2/6	PW3/7	
Resume Protection	Address	.5554					
	Data	..xxF5					

Figure 11 Overview of the Flash command sequence

Disable protection

- This command is to temporarily disable the password protection of the selected UCB (if UCB offers this feature) when all the passwords PW0-PW7 match their configured values in the corresponding UCB.

Resume protection

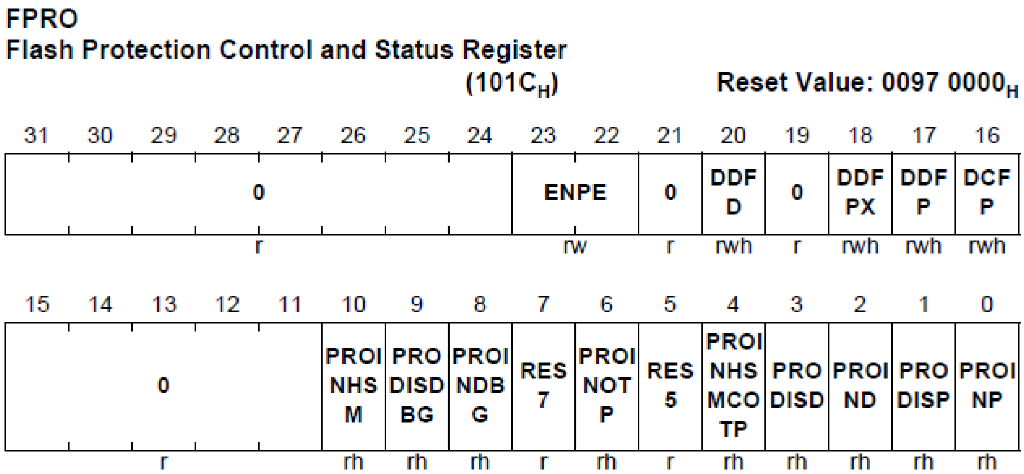
- This command is used to re-enable the Flash protection again, as it was configured.

In the section **4**, some examples are shown to give an idea on how to use these two commands during run-time.

1.5.2 Debug Protection Disable Status

The Flash Protection Control and Status Register reports the state of the Flash protection and contains protection relevant control fields.

FPRO.PRODISDBG is set to 1_B if a password is installed and the Flash command sequence 'Disable Protection' with matching password is applied.



PRODISDBG	9	rh	**Debug Interface Password Protection Disabled**[1)] The password configured by UCB_DBG was correctly received with "Disable Protection". When PROCONHSMCOTP.DESTDBG is "destructive" then only the SSW can disable this protection.

Figure 12 Flash Protection Status Register

The FPRO, Flash Protection Control and Status register has one bit related to the debug protection:

- Bit 9:
  - If read as 1_B, this bit indicates that the debug interface is unlocked, when the password matches.

1.6 SSW debug handling to unlock the debug interface

The Startup Software (SSW) is the first software executed after a chip reset.

One of the last device configuration steps performed by the SSW is the Debug System handling. This can set the SSW internal flag to unlock the debug interface if debug access to the device is enabled according to the following evaluation sequence:

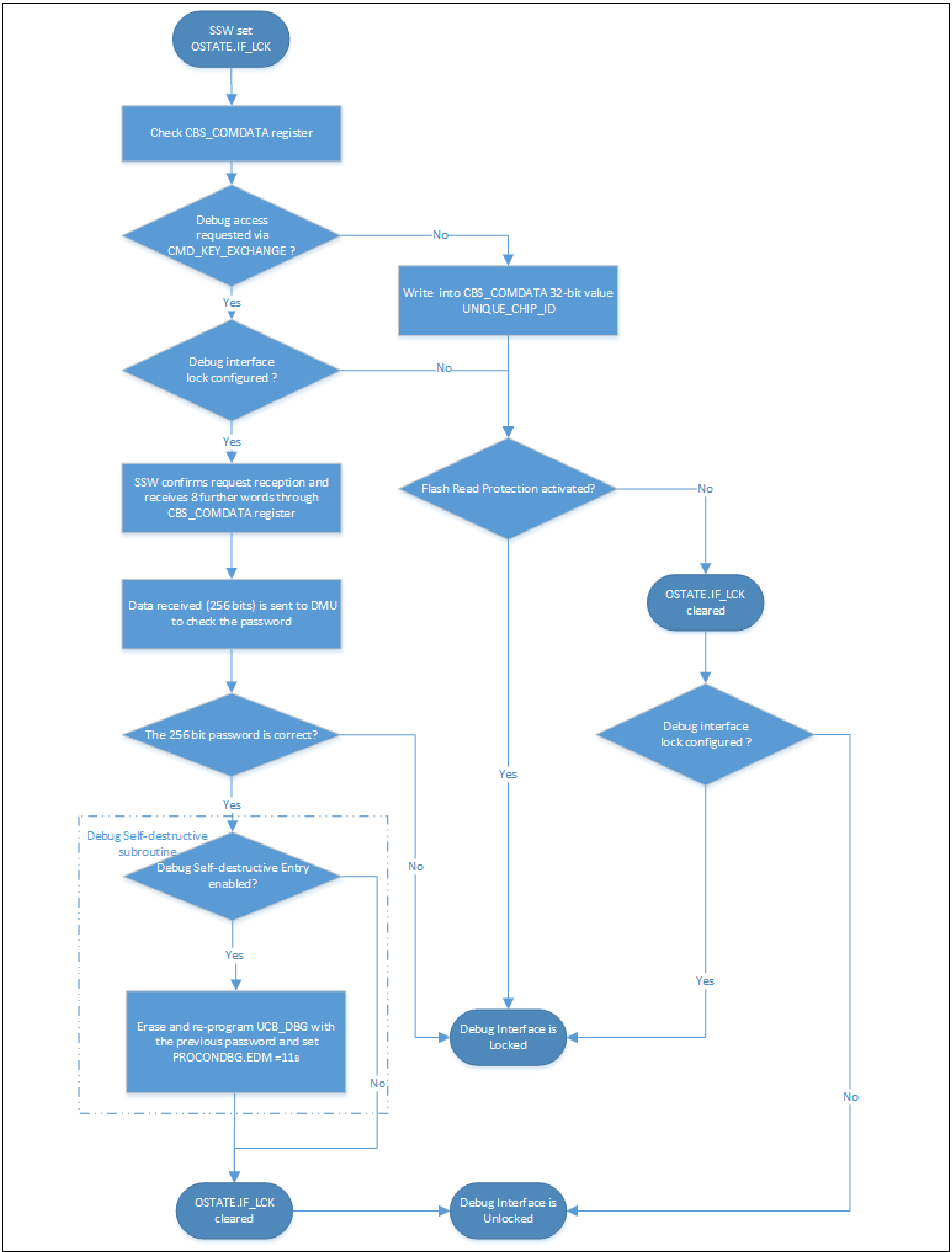


Figure 13 **Overview of debug handling in SSW**

1. Clear SSW internal flag "Debug allowed" (OSTATE.IF_LCK is set).
2. The SSW checks either an external tool has requested debug access by writing defined content (32-bit value CMD_KEY_EXCHANGE) into the CBS_COMDATA register.
   - CMD_KEY_EXCHANGE = 76D6E24AH for TC2xx
   - If the tool did not write the correct value in the CBS_COMDATA register, then the SSW will write the UNIQUE_CHIP_ID_32_BIT value in CBS_COMDATA to identify the device connected to the external tool.
3. The SSW checks either debug interface is configured as locked (PROCONDBG.DBGIFLCKis set).
4. Still in Cerberus Communication mode, the SSW confirms request receipt and receives 8 further words through the COMDATA register (256-bit debug password).
5. The Data received is sent by the SSW to the PMU to be checked as debug interface password, using the "Disable Protection" command sequence. The result is evaluated by the SSW.
6. Once the debug password is correct, check if the "Debug Self-destructive Entry" feature is activated for the device (PROCONHSMOTP.DESTDBG is11_B).
7. After debug access is granted, clear the SSW internal flag (OSTATE.IF_LCK). The debug interface will be unlocked. Exit the sequence.
8. If the debug interface lock is not configured and the Flash read protection is activated, the SSW internal flag (OSTATE.IF_LCK) remains set, the debug interface will be left locked and the sequence will be exited.
9. If the Flash read protection is not activated but the debug interface is configured as locked, the debug access will still remain locked (even though the SSW internal flag (OSTATE.IF_LCK) is cleared) as the PMU is blocking it by hardware.

This unlocking protocol can be supported by an external debugger tool, such as Infineon Memtool, PLS UDE, Lauterbach, iSystem and more.

Note:

1. *In the figure Overview of gates used to lock and unlocked the debug access, this protection layer is applicable by Gate 1 and Gate 2, where the SSW has the possibility to control the debug access.*
2. *More details of the evaluation sequence can be found in the AURIX™ TC2xx User Manual.*

2 Debug access protection use-cases

In order to cover all possible scenarios related to the debug interface mechanism, several possible use-cases to protect the debug access are described.

The following six use-cases demonstrate the high flexibility in terms of configurations applicable to the AURIX™ TC2xx device.

1. TriCore™ configure its own debug interface protection
2. TriCore™ configure its own debug interface protection, with Flash read protection
3. TriCore™ and HSM configure its own debug interface protection
4. TriCore™ and HSM configure its own debug interface protection, with Flash read protection
5. HSM configure both TriCore™ and HSM debug interface protection, with Flash read protection
6. HSM configure both TriCore™ and HSM debug interface protection, with debug password protection

2.1 Use-case 1: TriCore™ configure its own debug interface protection

This use-case covers the scenario when the protection is performed by the TriCore™ core only.

In order to apply this protection Gate 1 and Gate 2 (via PROCONDBG.OCDSDIS and PROCONDBG.DBGIFLCK and debug password) are taken into account, while Gate 3 and Gate 4 are kept unlocked.

Restricting debug access

The debug access is restricted using the following procedure:

- Disable the debug interface via UCB_DBG, setting it as described in the PROCONDBG register:
  - Set PROCONDBG.OCDSDIS = 1_B
  - Set PROCONDBG.DBGIFLCK = 1_B
  - Set the debug password to apply the protection mechanism.

The debug access is granted again only if an external entity provides the correct debug password.

Unlock sequence

For the unlock sequence, TriCore™ application code has to perform the following steps:

1. It needs a communication channel with the external world to receive the request for opening the debug interface and for exchanging authorization data (challenge and response mechanism).
   - (A) Communication can be via the TriCore™ SSW via the debug interface, as described in **1.6 SSW debug handling to unlock the debug interface.**
   - (B) This communication can be also done by TriCore™ via any other communication interface. For example, TriCore™ receives such a request via CAN (similar to the protocol used by the SSW).
   - In order to temporarily disable the OCDS protection, it needs to follow the command sequence 'Disable OCDS Protection' as described in Clear OCDS Interface Locked indication bit using software. This command sequence is done automatically by SSW (in case of part 1.A), and shall be done manually in application code (in case of part 1.B) in the event when the OCDSDIS bit is set to 1_B and the DBGIFLCK is clear to 0_B.

2.2 Use-case 2: TriCore™ configure its own debug interface protection, with Flash read protection

This use-case covers the scenario when the protection is performed by the TriCore™ core only, plus the PMU via the Read protection mechanism.

This protection is applied by Gate 1 and Gate 2, while Gate 3 and Gate 4 are kept unlocked.

Restricting debug access

The debug access is restricted using the same procedure:

- Disable the debug interface via UCB_DBG, setting it as described in the PROCONDBG register:
  - Set PROCONDBG.OCDSDIS = 1_B
  - Set PROCONDBG.DBGIFLCK = 1_B
  - Set the debug password to apply the protection mechanism

The PMU protection is applied using the Flash read protection mechanism:

- Disable the debug interface via UCB_PFlash (PROCONP0) or UCB_DFlash (PROCOND)
  - Set PROCONP0.RPRO = 1_B or PROCOND.RPRO = 1_B

The debug access is granted again only if an external entity provides the correct debug password and a piece of application code clears the OSTATE.IF_LCK bit.

Unlock sequence

For the unlock sequence, TriCore™ application code has to:

1. Establish a communication channel with the external world to receive the request for opening the debug interface and for exchanging authorization data (challenge and response mechanism).
   - (A) Communication can be via the TriCore™ SSW via the debug interface, as described in **1.6 SSW debug handling to unlock the debug interface.**
   - (B) This communication can be also done by TriCore™ via any other communication interface. For example, TriCore™ receives such a request via CAN (similar to the protocol used by the SSW).
   - In order to temporarily disable the OCDS protection, it needs to follow the command sequence 'Disable OCDS Protection' as described in **Clear OCDS Interface Locked indication bit using software**. This command sequence is done automatically by SSW (in case of part 1.A), and shall be done manually in application code (in case of part 1.B) in the event when the OCDSDIS bit is set to 1_B and the DBGIFLCK is clear to 0_B.
2. When TriCore™ accepted the request to open the debug interface then:
   - It releases the lock in OSTATE.IF_LCK by writing to OEC register with the correct pattern (as described in section **4**). This will be automatically handled in SSW, and shall be done manually in application code.

2.3 Use-case 3: TriCore™ and HSM configure their own debug access protection separately

This use-case covers the scenario when the debug protection is applied by both the TriCore™ and HSM sides, but each one only for itself.

This protection is applied by Gate 1 and Gate 2 and Gate 4 (adding the UCB_HSM protection via PROCONHSM.HSMDBDIS), while Gate 3 is kept unlocked.

Restricting debug access - TriCore™

The debug access is restricted as follows:

- Disable the debug interface via UCB_DBG, setting it as described in the PROCONDBG register:
  - Set PROCONDBG.OCDSDIS = 1_B
  - Set PROCONDBG.DBGIFLCK = 1_B
  - Set the debug password to apply the protection mechanism

Restricting debug access - HSM

For the HSM side, the debug access is locked via the following procedure:

- Lock the debug access via UCB_HSM, setting it as described in the PROCONHSM register:
  - Set PROCONHSM.HSMDBGDIS = 1_B
  - Set PROCONHSM.DBGIFLCK = 0_B (This keeps the Gate 3 unlocked)

The debug access to TriCore™ is granted again only if an external entity provides the correct debug password.

In order to also access the HSM core, the HSM has to unlock Gate 4.

Unlock sequence

For the unlock sequence, TriCore™ application code has to perform step 1 to open the debug access. The additional steps to unlock the HSM debug access have to be performed by the HSM application code:

1. Establish a communication channel with the external world to receive the request for opening the debug interface and for exchanging authorization data (challenge and response mechanism).
   - (A) Communication can be via the TriCore™ SSW via the debug interface, as described in **1.6 SSW debug handling to unlock the debug interface.**
   - (B) This communication can be also done by TriCore™ via any other communication interface. For example, TriCore™ receives such a request via CAN (similar to the protocol used by the SSW).
   - In order to temporarily disable the OCDS protection, it needs to follow the command sequence 'Disable OCDS Protection' as described in **Clear OCDS Interface Locked indication bit using software.** This command sequence is done automatically by SSW (in case of part 1.A), and shall be done manually in application code (in case of part 1.B) in the event when the OCDSDIS bit is set to 1_B and the DBGIFLCK is clear to 0_B.
2. The HSM application code needs a communication channel with the external world to receive the request for opening the debug interface and for exchanging authorization data (challenge and response mechanism).
   - This communication can be done by HSM over the OCDS COMDATA register.
   - This communication can be also done via any other communication interface. For example, TriCore™ receives such a request via CAN and forwards it to HSM for checking.

3. When HSM accepts also a request to open HSM debug interface then:
   - HSM code releases the HSM debug lock by changing the DBGCTRL.HSM register to 11_B.

2.4 Use-case 4: TriCore™ and HSM configure their own debug access protection, with Flash read protection

This use-case covers the scenario when the debug protection is applied by both the TriCore™ and HSM sides, but each one only for itself, and also the PMU module via the Flash Read protection mechanism.

This protection is applied by Gate 1, Gate 2 and Gate 4 (adding UCB_PFLASH and UCB_DFLASH protection via PROCONP0.RPRO and PROCOND.RPRO bit), while Gate 3 is kept unlocked.

Restricting debug access - TriCore™

The debug access is restricted with procedure:

- Disable the debug interface via UCB_DBG, setting it as described in the PROCONDBG register:
  - Set PROCONDBG.OCDSDIS = 1_B
  - Set PROCONDBG.DBGIFLCK = 1_B
  - Set the debug password to apply the protection mechanism

Restricting debug access - HSM

For the HSM side, the debug access is locked via the following procedure:

- Lock the debug access via UCB_HSM, setting it as described in the PROCONHSM register:
  - Set PROCONHSM.HSMDBGDIS = 1_B
  - Set PROCONHSM.DBGIFLCK = 0_B (This keeps the Gate 3 unlocked)

The PMU protection is applied using the Flash read protection mechanism:

- Disable the debug interface via UCB_PFlash (PROCONP0) or UCB_DFlash (PROCOND)
  - Set PROCONP0.RPRO = 1_B or PROCOND.RPRO = 1_B

The debug access to TriCore™ is granted again only if an external entity provides the correct debug password and a piece of application code clears the OSTATE.IF_LCK bit.

In order to be able to access also the HSM core, the HSM has to unlock Gate 4.

Unlock sequence

For the unlock sequence, TriCore™ application code and HSM application code have to perform the same steps as described in the previous use-case to open both debug interfaces. Some additional steps in TriCore™ application code have to be taken into account.

1. Establish a communication channel with the external world to receive the request for opening the debug interface and for exchanging authorization data (challenge and response mechanism).
   - (A) Communication can be via the TriCore™ SSW via the debug interface, as described in **1.6** SSW debug handling to unlock the debug interface**.**
   - (B) This communication can be also done by TriCore™ via any other communication interface. For example, TriCore™ receives such a request via CAN (similar to the protocol used by the SSW).
   - In order to temporarily disable the OCDS protection, it needs to follow the command sequence 'Disable OCDS Protection' as described in Clear OCDS Interface Locked indication bit using software. This command sequence is done automatically by SSW (in case of part 1.A), and shall be done manually in application code (in case of part 1.B) in the event when the OCDSDIS bit is set to 1_B and the DBGIFLCK is clear to 0_B.

2. When TriCore™ accepts the request to open the TriCore™ debug interface then:
   - TriCore™ code releases the lock in OSTATE.IF_LCK by writing to the CBS_OEC register.
3. When HSM accepts also a request to open HSM debug interface then:
   - HSM code releases the HSM debug lock by changing the DBGCTRL.HSM register to 11_B.

2.5 Use-case 5: HSM configures both TriCore™ and HSM Debug access protection, with Flash read protection

This use-case covers the scenario when the debug protection to the entire device is applied by HSM only and optionally also by the PMU module.

This protection is applied via Gate 3 and Gate 4 while Gate 1 and Gate 2 are kept unlocked.

Note: Gate 2 can be locked if Flash read protection is enabled, adding another protection layer.

Restricting debug access

Debug access to TriCore™ and to HSM is restricted using only the UCB_HSM:

- Disable the debug access and lock the HSM debug support via UCB_HSM (PROCONHSM)
  - Set PROCONHSM.HSMDBGDIS = 1_B (to close the Gate 4)
  - Set PROCONHSM.DBGIFLCK = 1_B (to close the Gate 3)
- Optionally, disable the debug interface via UCB_PFlash (PROCONP0) or UCB_DFlash (PROCOND)
  - Set PROCONP0.RPRO = 1_B or PROCOND.RPRO = 1_B (to close the Gate 2)

Re-enable debug access

The HSM application code can re-enable debug access using the DBGCTRL register:

- Set DBGCTRL.HSM = 11_B, HSM debug enabled
- Set DBGCTRL.HOST = 11_B, Host debug enabled

The debug access is granted again only if an external entity provides the correct challenge and response password to the HSM Core to unlock it and a piece of application code clears the OSTATE.IF_LCK bit, if read protection is applied.

Unlock sequence

For the unlock sequence, the HSM application code has to follow these steps:

1. A communication channel with the external world to receive the request for opening the debug interface and for exchanging authorization data (challenge and response mechanism).
   - Communication can be via the HSM over the OCDS COMDATA register (similar to the protocol used by the SSW).
   - Communication can be also be made via any other communication interface. For example, TriCore™ receives such requests via CAN and forwards it to HSM for checking.
2. When HSM accepted the request to open the TriCore™ debug interface then:
   - It releases its own lock of the debug interface by changing DBGCTRL.HOST to 11_B.
   - It releases the lock in OSTATE.IF_LCK by writing to CBS_OEC register, if read protection is applied.
3. When HSM has accepted a request to open HSM debug interface then:
   - It releases the HSM debug lock by changing DBGCTRL.HSM to 11_B.

2.6 Use-case 6: HSM configures both TriCore™ and HSM debug access protection, with Debug password protection

This use-case covers the scenario when the debug protection to the entire device is applied by only by HSM, and it also applies the debug password protection mechanism.

This protection is applied via Gate 1, Gate 2, Gate 3 and Gate 4.

Restricting debug access

The debug access to TriCore™ and the HSM is restricted using the UCB_HSM.

Debug password protection is applied using UCB_DBG:

- Lock the HSM debug support via UCB_HSM (PROCONHSM)
  - Set PROCONHSM.HSMDBGDIS = 1_B
  - Set PROCONHSM.DBGIFLCK = 0_B
- Disable the debug access with debug password via UCB_DBG (PROCONDBG)
  - Set PROCONDBG.OCDSDIS = 1_B
  - Set PROCONDBG.DBGIFLCK = 1_B
  - Set the debug password to apply the protection mechanism

Unlock sequence

For the unlock sequence, the HSM application code has to perform the same steps as described in the previous use-case to re-enable the debug access. An additional step in the HSM application code has to be taken into account to unlock the debug password protection mechanism:

1. A communication channel with the external world is required to receive the request for opening the debug interface and for exchanging authorization data (challenge and response mechanism).
   - Communication can be via the HSM over the OCDS COMDATA register (similar to the protocol used by the SSW)
   - Communication can be also be made via any other communication interface. For example, TriCore™ receives such requests via CAN and forwards it to HSM for checking
2. When HSM accepted the request to open the TriCore™ debug interface then:
   - It releases its own lock of the debug interface by changing DBGCTRL.HOST to 11_B
   - It releases the lock in OSTATE.IF_LCK by writing to CBS_OEC register, if read protection is applied.
3. When HSM has accepted a request to open HSM debug interface then:
   - It releases the HSM debug lock by changing DBGCTRL.HSM to 11_B
4. Provide the password by using the command sequence "Disable Protection".
   - HSM needs to know the correct password for this.
   - One solution is that this Debug password is stored in the HSM Data Flash.

3 Lock/unlock the debug access using debugger tools

The SSW debug handling protocol to unlock the debug access can be supported by an external debugger tool, such as Memtool, PLS UDE, Lauterbach or iSystem. It can be also supported by software implemented in application code. The procedure to follow has been described in section **1.4.1**.

The following figure gives a memory view of the content of UCB_DBG at location 0xAF101400.

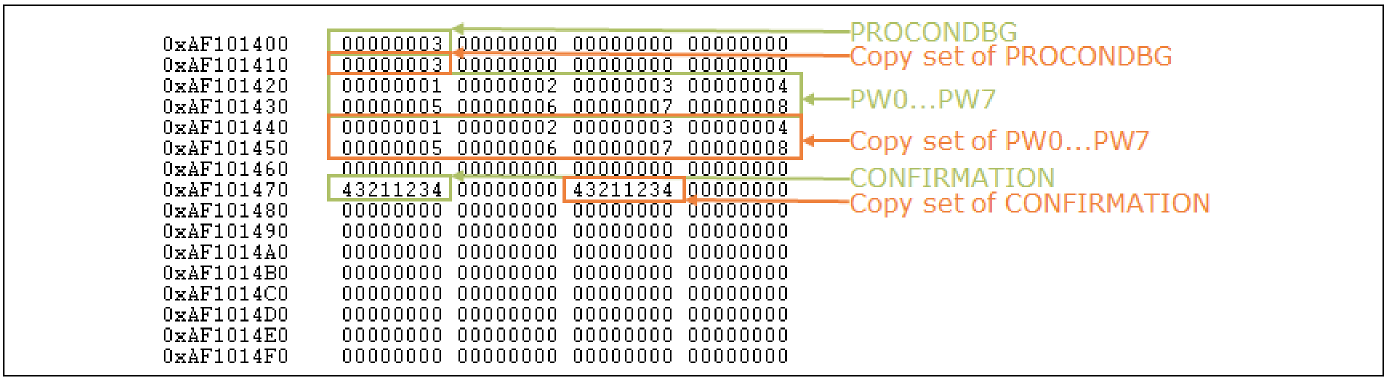


Figure 14 Memory view of the UCB_DBG

3.1 Lock using the Infineon Memtool/PLS UDE Memtool

In the Memtool, a user interface enables you to easily modify the UCB_DBG and UCB_HSM content:

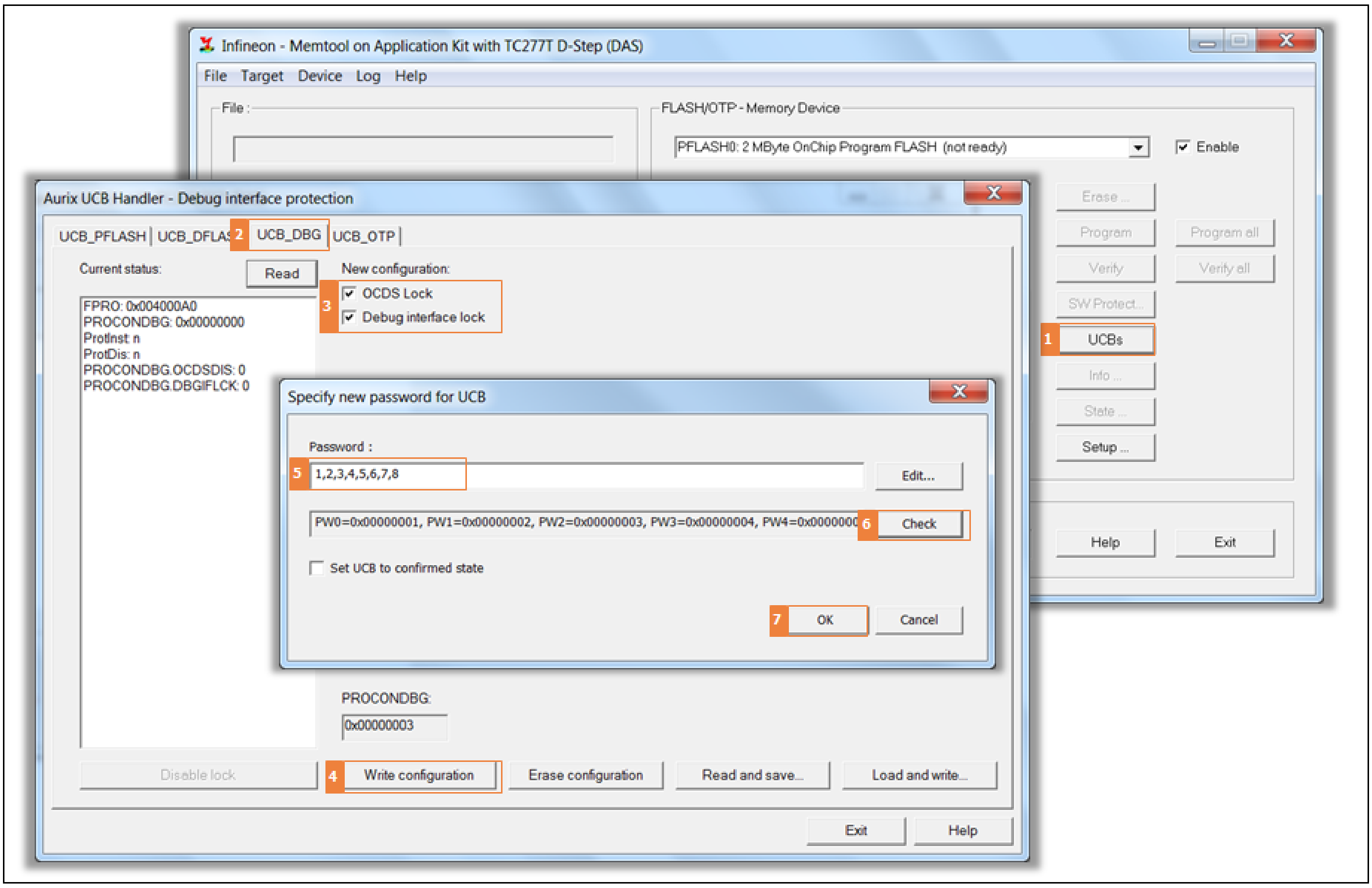


Figure 15 UCB_DBG configuration for Lock debug access with password via Memtool

Lock/unlock the debug access using debugger tools

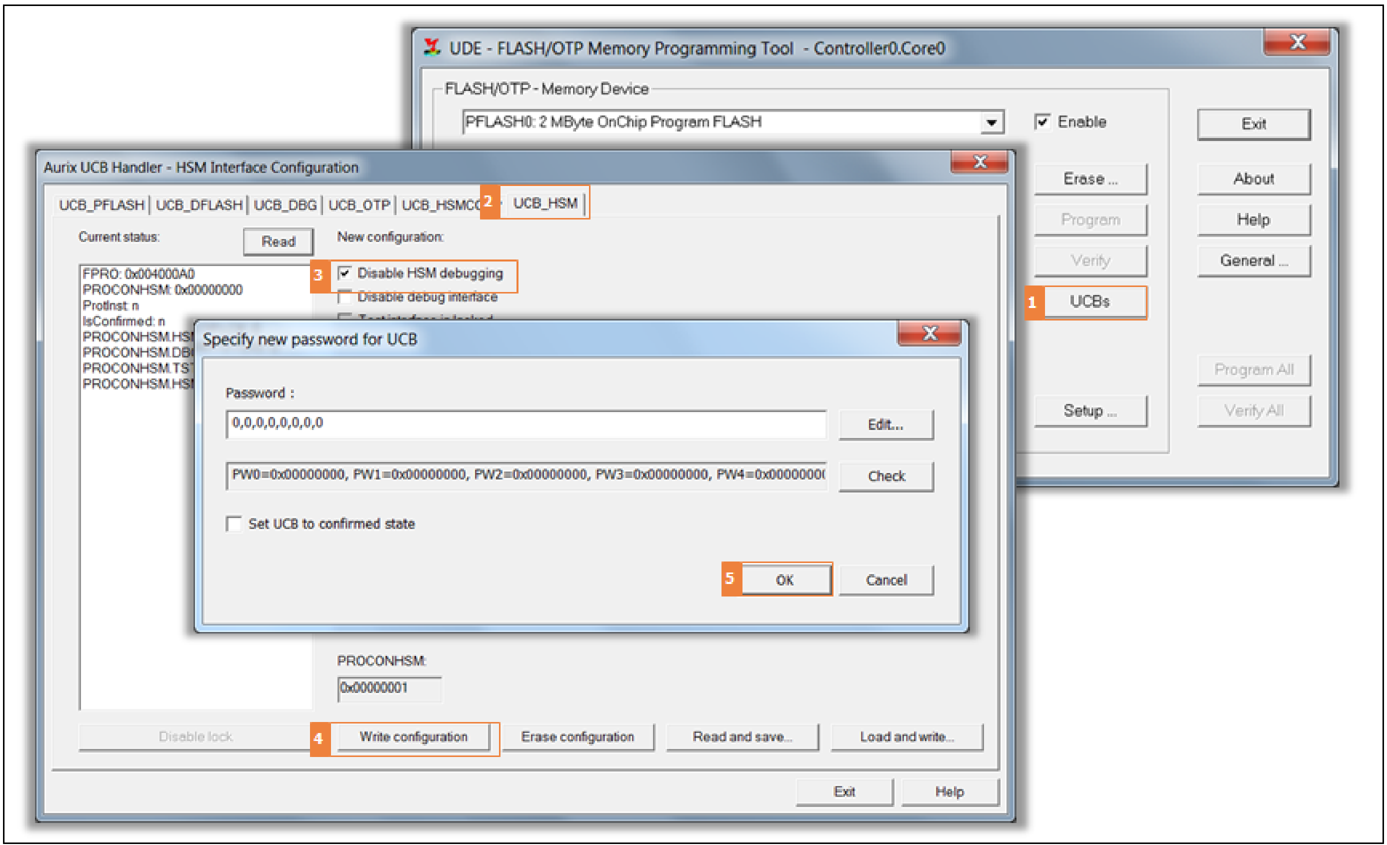


Figure 16 UCB_HSM configuration for HSM debug disabled via Memtool

3.2 Unlock using the Infineon Memtool/PLS UDE Memtool

In order to provide the correct password to the UCB_DBG, the following user interface can be used:

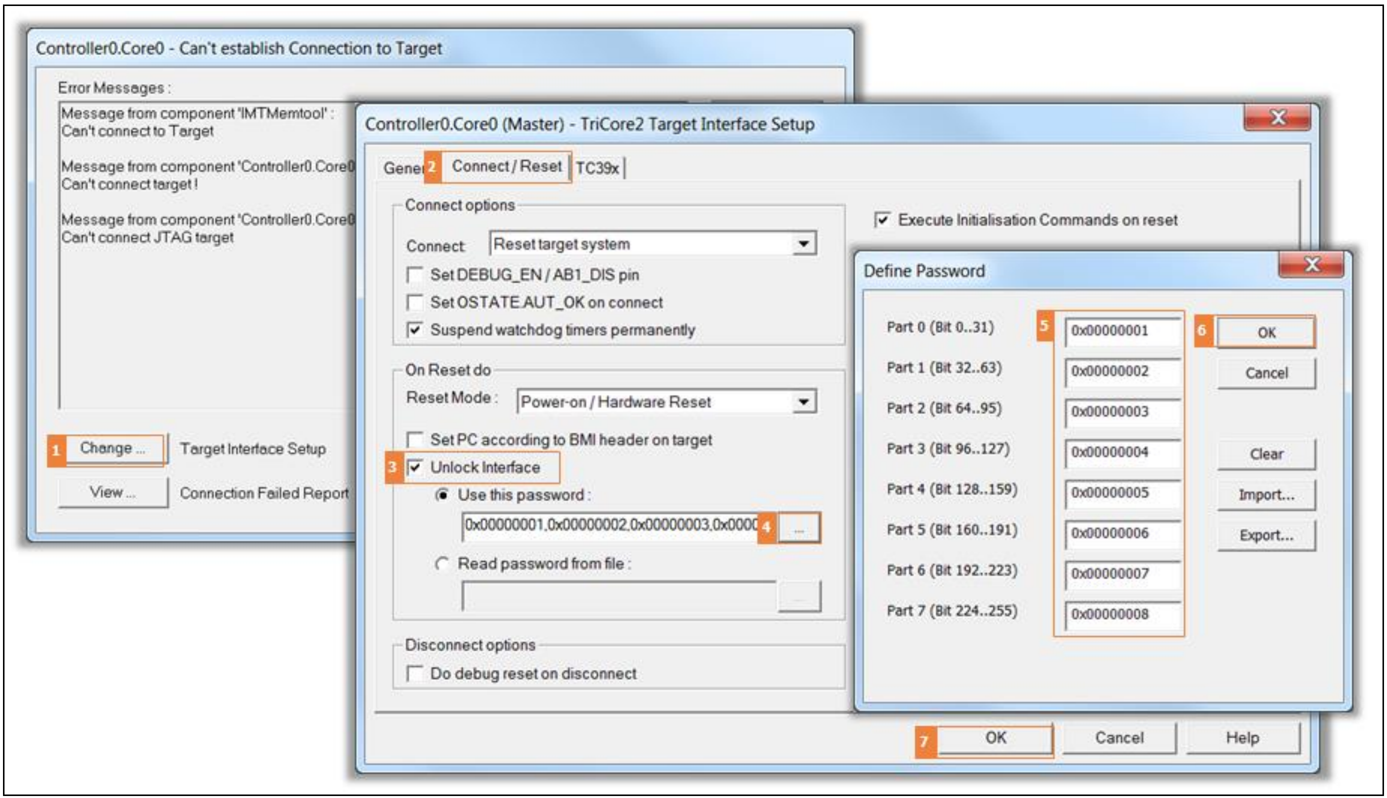


Figure 17 UCB_DBG configuration for Unlock debug access with password via Memtool

3.3 Lock using Lauterbach

An alternative way to set the UCB_DBG password using the Lauterbach debugger is by using a script.

The following extracted code sets the:

- PROCONDBG.OCDSDIS bit to 1
- PROCONDBG.DBGIFLCK bit to 1
- user defined password in the UCB

Code Listing 1

```
;Example for download
DIALOG.YESNO "Program UCB_DBG for password protection?"
LOCAL &progflash
ENTRY &progflash
IF &progflash
(
   ;Enable flash programming
   FLASH.AUTO 0xaf101400--0xaf1017ff     ;UCB_DBG
   Data.Set 0xaf101400 %Long 0x00000003 ;Set PROCONDBG value
   Data.Set 0xaf101410 %Long 0x00000003 ;Copy set of PROCONDBG value
   Data.Set 0xaf101420 %Long 0x00000001 ;Set Pwd0(Bit 0..31)
   Data.Set 0xaf101424 %Long 0x00000002 ;Set Pwd1(Bit 32..63)
```

Code Listing 1

```
            Data.Set 0xaf101428 %Long 0x00000003 ;Set Pwd2(Bit 64..95)
            Data.Set 0xaf10142C %Long 0x00000004 ;Set Pwd3(Bit 96..127)
            Data.Set 0xaf101430 %Long 0x00000005 ;Set Pwd4(Bit 128..159)
            Data.Set 0xaf101434 %Long 0x00000006 ;Set Pwd5(Bit 160..191)
            Data.Set 0xaf101438 %Long 0x00000007 ;Set Pwd6(Bit 192..223)
            Data.Set 0xaf10143C %Long 0x00000008 ;Set Pwd7(Bit 224..255)
            Data.Set 0xaf101440 %Long 0x00000001 ;Copy set of Pwd0
            Data.Set 0xaf101444 %Long 0x00000002 ;Copy set of Pwd1
            Data.Set 0xaf101448 %Long 0x00000003 ;Copy set of Pwd2
            Data.Set 0xaf10144C %Long 0x00000004 ;Copy set of Pwd3
            Data.Set 0xaf101450 %Long 0x00000005 ;Copy set of Pwd4
            Data.Set 0xaf101454 %Long 0x00000006 ;Copy set of Pwd5
            Data.Set 0xaf101458 %Long 0x00000007 ;Copy set of Pwd6
            Data.Set 0xaf10145C %Long 0x00000008 ;Copy set of Pwd7
            Data.Set 0xaf101470 %Long 0x43211234 ;Set confirmation value
            Data.Set 0xaf101478 %Long 0x43211234 ;Copy set of
  confirmation value
            FLASH.AUTO off
         )
         ENDDO
```

3.4 Unlock using Lauterbach

In order to provide the debug password to unlock the debug interface, Lauterbach provides the following command:

Format: **SYStem.Option KEYCODE** [<pwd0> <pwd1> ... <pwd7>]

SYStem.Option KEYCODE 0x00000001 0x00000002 0x00000003 0x00000004 0x00000005 0x00000006 0x00000007 0x00000008

Figure 18 Unlock debug access with debug password via the Lauterbach script

3.5 Locking using iSystem

Using the iSystem debugger, a python script can be used to lock the debug access.

The following extracted code sets the:

- PROCONDBG.OCDSDIS bit to 1
- PROCONDBG.DBGIFLCK bit to 1
- user defined password in the UCB

Code Listing 2

```
import isystem.connect as ic

cmgr = ic.ConnectionMgr()
cmgr.connectMRU('')
ideCtrl = ic.CIDEController(cmgr)
dataCtrl = ic.CDataController(cmgr)
```

Code Listing 2

```

        memoryAreas = dataCtrl.getSystemMemoryAreas()
        maPhysical = memoryAreas.getMemAreaDataPhysical()

        valueType = ic.SType()
        valueType.m_byType = ic.SType.tUnsigned
        valueType.m_byBitSize = 32

        # Set PROCONDBG
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101400, ic.CValueType(valueType, 0x00000003))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101410, ic.CValueType(valueType, 0x00000003))

        # Set PW0-PW7
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101420, ic.CValueType(valueType, 0x00000001))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101424, ic.CValueType(valueType, 0x00000002))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101428, ic.CValueType(valueType, 0x00000003))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF10142C, ic.CValueType(valueType, 0x00000004))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101430, ic.CValueType(valueType, 0x00000005))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101434, ic.CValueType(valueType, 0x00000006))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101438, ic.CValueType(valueType, 0x00000007))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF10143C, ic.CValueType(valueType, 0x00000008))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101440, ic.CValueType(valueType, 0x00000001))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101444, ic.CValueType(valueType, 0x00000002))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101448, ic.CValueType(valueType, 0x00000003))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF10144C, ic.CValueType(valueType, 0x00000004))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101450, ic.CValueType(valueType, 0x00000005))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101454, ic.CValueType(valueType, 0x00000006))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101458, ic.CValueType(valueType, 0x00000007))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF10145C, ic.CValueType(valueType, 0x00000008))

        # Set Confirmation value
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101470, ic.CValueType(valueType, 0x43211234))
        dataCtrl.writeValue(ic.IConnectDebug.fRealTime, maPhysical, 0xAF101478, ic.CValueType(valueType, 0x43211234))

```

Code Listing 2

```
ideCtrl.refreshUI()
```

In order to execute this code, choose Tools and then Run Script option, as described below:

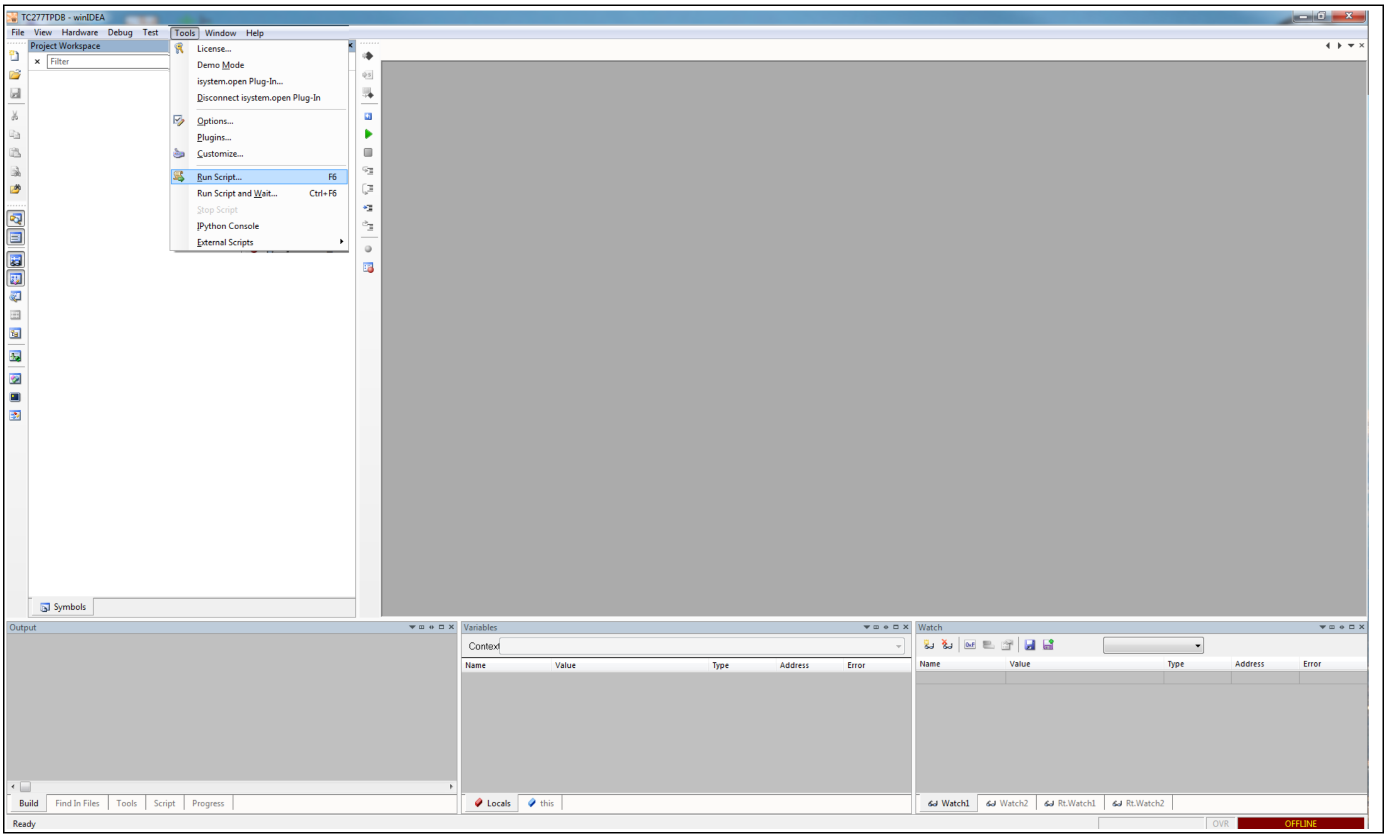


Figure 19 UCB_DBG configuration with iSystem script execution

3.6 Unlocking using iSystem

In order to provide the debug password to unlock the debug access, iSystem provides the following method from the CPU Setup:

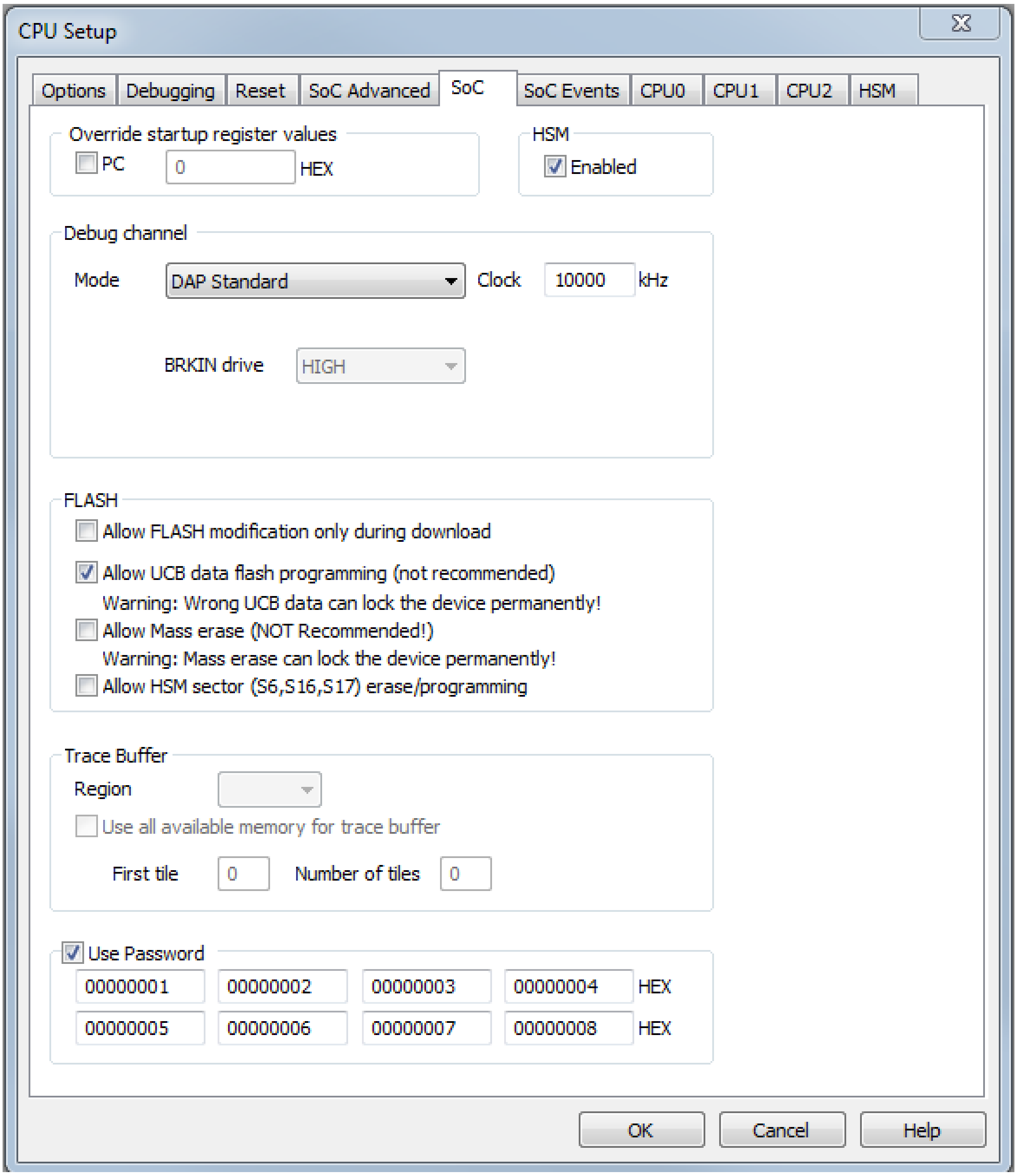


Figure 20 Unlock debug access with debug password via iSystem GUI

4 Lock/unlock debug interface using software

4.1 Enable debug access password protection using software

In order to program the UCB_DBG, an extracted example code is provided here.

This example code sets:

- The PROCONDBG.OCDSDIS bit to 1 to disable OCDS
- The PROCONDBG.DBGIFLCK bit to 1 to lock the debug interface with a user defined password in the UCB_DBG

Code Listing 3

```
void flash_setUcbDbg(void)
{
  uint32 i;
  uint32 buffer[0x20];                    //128 Byte
  uint32 addr = 0xAF101400;               //UCB_DBG

  flash_eraseDflash(addr, 1);             //erase DF_UCB
  while ((FLASH0_FSR.U & 0x1E) != 0) ;  //Wait Flash is not busy

  for (i = 0; i < 0x20; i += 1)
    buffer[i] = 0x0;                      //clear buffer

  buffer[0x0]  = 0x00000003;        //Set PROCONDBG 1st Set
  buffer[0x4]  = 0x00000003;        //Copy of PROCONDBG
  buffer[0x8]  = 0x00000001;        //Set 256-bit Password 0
  buffer[0x9]  = 0x00000002;        //Set 256-bit Password 1
  buffer[0xA]  = 0x00000003;        //Set 256-bit Password 2
  buffer[0xB]  = 0x00000004;        //Set 256-bit Password 3
  buffer[0xC]  = 0x00000005;        //Set 256-bit Password 4
  buffer[0xD]  = 0x00000006;        //Set 256-bit Password 5
  buffer[0xE]  = 0x00000007;        //Set 256-bit Password 6
  buffer[0xF]  = 0x00000008;        //Set 256-bit Password 7
  buffer[0x10] = 0x00000001;        //Copy of 256-bit Password 0
  buffer[0x11] = 0x00000002;        //Copy of 256-bit Password 1
  buffer[0x12] = 0x00000003;        //Copy of 256-bit Password 2
  buffer[0x13] = 0x00000004;        //Copy of 256-bit Password 3
  buffer[0x14] = 0x00000005;        //Copy of 256-bit Password 4
  buffer[0x15] = 0x00000006;        //Copy of 256-bit Password 5
  buffer[0x16] = 0x00000007;        //Copy of 256-bit Password 6
  buffer[0x17] = 0x00000008;        //Copy of 256-bit Password 7
  buffer[0x1C] = 0x43211234;        //Set Confirmation code
  buffer[0x1E] = 0x43211234;        //Copy of Confirmation code

  for (i = 0; i < (0x20/8); i += 1) {
    flash_programDflash(addr + i * 32, &buffer[i * 8]); //program DF_UCB
    while ((FLASH0_FSR.U & 0x1E) != 0) ; //Wait Flash is not busy (DF0, DF1, PF0 and PF1)
  }
}
```

Code Listing 4

```
uint32 flash_eraseDflash(uint32 addr, uint32 cnt)
{
  if (!cnt)
    return (1);      //Stop operation if no sector

  //Erase sector
  *(volatile uint32 *)(0xAF00AA50) = (uint32) addr;
  *(volatile uint32 *)(0xAF00AA58) = (uint32) cnt;
  *(volatile uint32 *)(0xAF00AAA8) = (uint32) 0x80;
  *(volatile uint32 *)(0xAF00AAA8) = (uint32) 0x50;
  __dsync();

  return (0);
}
```

Code Listing 5

```
uint32 flash_programDflash(uint32 addr, uint32* pmem)
{
  volatile uint32 i;
  volatile uint32 low32bit, high32bit;

  //Enter page mode
  *(volatile uint32 *)(0xAF005554) = 0x5D;
  __dsync();

  if ( ((FLASH0_FSR.U & 0x0400) == 0) || ((FLASH)_FSR.U & 0x3800) != 0) )
    return (1);      //Stop operation if error

  i = 0;
  while (i < 0x8)    //Load page
  {
    low32bit = pmem[i];
    high32bit = pmem[i + 1];
    *(volatile uint32 *)(0xAF0055F0) = low32bit;
    *(volatile uint32 *)(0xAF0055F4) = high32bit;
  __dsync();
    i += 2;
  }

  //Write burst
  *(volatile uint32 *)(0xAF00AA50) = (uint32) addr;
  *(volatile uint32 *)(0xAF00AA58) = (uint32) 0x00;
  *(volatile uint32 *)(0xAF00AAA8) = (uint32) 0xA0;
  *(volatile uint32 *)(0xAF00AAA8) = (uint32) 0x7A;
  __dsync();

  return (0);
}
```

4.2 Disable protection mechanism for UCB_DBG using software

The following example code describes how to use the UCB command sequence to temporarily disable the protection of the UCB by providing the correct debug password.

Code Listing 6

```
void ucb_disableProtection(void)
{
  //Reset to read
  *(volatile uint32 *)(0xAF005554) = 0x000000F0;

  //Disable protection
  *(volatile uint32 *)(0xAF00553C) = 0x00000005;    //UCB_DBG
  *(volatile uint32 *)(0xAF00553C) = 0x00000001;    //PW0
  *(volatile uint32 *)(0xAF00553C) = 0x00000002;    //PW1
  *(volatile uint32 *)(0xAF00553C) = 0x00000003;    //PW2
  *(volatile uint32 *)(0xAF00553C) = 0x00000004;    //PW3
  *(volatile uint32 *)(0xAF00553C) = 0x00000005;    //PW4
  *(volatile uint32 *)(0xAF00553C) = 0x00000006;    //PW5
  *(volatile uint32 *)(0xAF00553C) = 0x00000007;    //PW6
  *(volatile uint32 *)(0xAF00553C) = 0x00000008;    //PW7

  while ((FLASH0_FSR.U & 0x1E) != 0) ;   //Wait Flash is not busy (DF0, DF1, PF0 and PF1)
}
```

4.3 Resume protection mechanism for UCB_DBG using software

The following example code shows how to use the UCB command sequence to resume the protection of the UCB.

Code Listing 7

```
void ucb_resumeProtection(void)
{
  //Reset to read
  *(volatile uint32 *)(0xAF005554) = 0x000000F0;

  //Resume protection
  *(volatile uint32 *)(0xAF005554) = 0x000000F5;

  while ((FLASH0_FSR.U & 0x1E) != 0) ;   //Wait Flash is not busy (DF0, DF1, PF0 and PF1)
}
```

4.4 Clear OCDS Interface Locked indication bit using software

This example code describes how to unlock the debug interface from application code running from the TriCore™ or HSM core. This code can be also used in order to unlock the debug interface once the Flash read protection configuration bit, PROCONP0.RPRO or PROCOND.RPRO has been applied.

Code Listing 8

```
void ocds_clearInterfaceLocked(void)
{
```

Code Listing 8

```
      //OCDS enabling pattern (access needs to be done at 32 bit)
      CBS_OEC.U = 0xA1;
      CBS_OEC.U = 0x5E;
      CBS_OEC.U = 0xA1;
      CBS_OEC.U = 0x5E;

      //Clear OSTATE.IF_LCK with write access enabled
      CBS_OEC.U = 0x00010000;
      //Wait debug interface is unlocked
      while ((OSTATE.B.IF_LCK) != 0) ;
    }
```

Revision history

Document version	Date of release	Description of changes
V 1.0	2017-10	Initial Version
V1.1	2018-06	Updated all sections. Added iSystem tool support.
V1.2	2019-06	Added the effect of an open HSM debug interface in Section 1.4.3.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2019-06

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32389